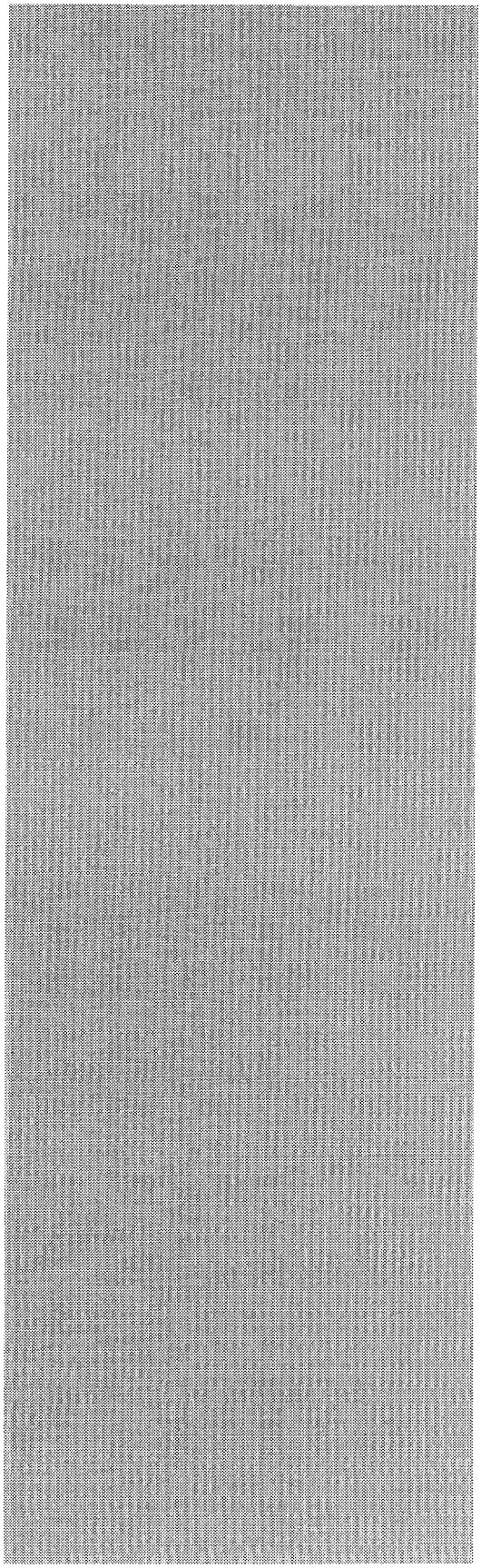
LA-13032



Studies of Parallel Algorithms for the Solution of a Fokker-Planck Equation

Los Alamos
NATIONAL LABORATORY

Los Alamos National Laboratory is operated by the University of California for the United States Department of Energy under contract W-7405-ENG-36.

Edited by Patricia W. *Mendius,* ***Group*** *CIC-1*
Prepared by Ann *Nagy,* ***Group*** *XTM*

LA-13032

UC-712
Issued: November 1995

Studies of Parallel Algorithms for the Solution of a Fokker-Planck Equation

*Dominique Deck**
*Gérald Samba***

"Visiting Scientist. Temporary address: Applied Theoretical& Computational Physics Division, Los Alamos, National Laboratory, Los Alamos, NM 87545
*'*Visiting Scientist, Département de mathématiques appliquées, CEA/CEL-V, 94295 Villeneuve st Georges CEDEX, FRANCE.*

Los Alamos
NATIONAL LABORATORY
Los Alamos, New Mexico 87545

STUDIES OF PARALLEL ALGORITHMS FOR THE SOLUTION OF A FOKKER-PLANCK EQUATION

by

Dominique Deck* and Gérald Samba**

ABSTRACT

The study of laser-created plasmas often requires the; use of a kinetic model rather than a hydrodynamic one. This model change occurs, for example, in the hot spot formation in an ICF experiment or during the relaxation of colliding plasmas. When the gradients scale-lengths or the size of a given system are not small compared to the characteristic mean-free-path, we have to deal with non-equilibrium situations, which can be described by the distribution functions of every species in the system. We present here a numerical method in plane or spherical 1-D geometry, for the solution of a Fokker-Planck equation that describes the evolution of such functions in the phase space. The size and the time scale of kinetic simulations require the use of Massively Parallel Computers (MPP). We have adopted a message-passing strategy using Parallel Virtual Machine (PVM).

* Temporary address: Applied Theoretical & Computational Physics Division, Los Alamos National Laboratory, Los Alamos, NM 87545

** Département d'mathématiques appliquées, CEA/CEL-V, 94195 Villeneuve st Georges CEDEX, FRANCE

I. PHYSICAL MODEL

A. Hypothesis and Equations

We consider the ions of the plasma as nonequilibrium populations interacting both all together and with the electrons. We make no approximation on the shape of the ion distribution functions. We only take into account the far coulomb interactions which are well described by the Fokker-Planck collision operator.

1. Approximations

The smallest time-scale we are interested in is the time τ_{ii} of ion-ion collision. The electron-electron collision time is ϵ times smaller than τ_{ii}, where $\epsilon = \sqrt{\frac{m_e}{m_i}}$, m_e and m_i being respectively the electron mass and the ion mass. We assume that on a time τ_{ii}, the electrons are at a Maxwellian equilibrium. We replace the electronic Fokker-Planck equation by a diffusion equation for their internal energy.

In all our applications, the Debye length λ_d, is very small compared to the gradient scale lengths and to the collision mean free path. So we assume quasi-neutrality of the plasma.

2. Equations

The Fokker-Planck equation for any particle species, say α, reads:

$$\frac{\partial f_\alpha}{\partial t} + v_i \frac{\partial f_\alpha}{\partial x_i} + \frac{q_\alpha E_i}{m_\alpha} \frac{\partial f_\alpha}{\partial v_i} = \sum_\beta Q(f_\alpha, \mathbf{f},) \ .$$

where $\vec{E}$ is the electric field, q_α the charge of the particle α, m_α its mass, f_α its distribution function, and $Q(f_\alpha, f_\beta)$ the Fokker-Planck collision operator.

We will consider plane or spherical geometries. The Fokker-Planck equation will be solved in a moving frame in order to deal correctly with a "piston" (pusher in ICF). The transport part of the equation is the only one to be modified by this change of frame. We also normalize the ionic velocity with a selected one $v_0(r,t)$. The space dependency of v_0 guarantees that we will be able to discretize a steep temperature gradient well. The time dependence of v_0 allows us to keep this accuracy during the simulation.

Taking into account these transformations in 1-D spherical geometry, the equation now reads:

$$\begin{aligned}
&\frac{\partial (r^2 v_\perp f_\alpha)}{\partial t} + \frac{\partial (r^2 v_\perp \dot{r} f_\alpha)}{\partial r} + \frac{\partial (r^2 v_\perp v_\parallel v_0 f_\alpha)}{\partial r} = \sum_\beta Q(f_\alpha, f_\beta) \\
&+ \frac{\partial}{\partial v_\parallel} \left[r^2 v_\perp \left(-\frac{1}{r} v_\perp^2 v_0 - \frac{q_\alpha E}{m_\alpha v_0} + \frac{\ddot{r}}{v_0} + v_\parallel \frac{\partial \dot{r}}{\partial r} + \frac{v_\parallel}{v_0} \frac{dv_0}{dt} + v_\parallel^2 \frac{\partial v_0}{\partial r} \right) f_\alpha \right] \qquad (1.1) \\
&+ \frac{\partial}{\partial v_\perp} \left[r^2 v_\perp^2 \left(\frac{1}{r} v_\parallel v_0 + \frac{\dot{r}}{r} + \frac{1}{v_0} \frac{dv_0}{dt} + v_\parallel \frac{\partial v_0}{\partial r} \right) f_\alpha \right] \ .
\end{aligned}$$

where $\dot{r}$ is a grid velocity , $\ddot{r}$ its acceleration. We assume a cylindrical symmetry of f_α in velocity space.

The equation for the electron temperature is

$$\frac{3}{2}\left(\frac{\partial T_e}{\partial t} + u_{ei}\frac{\partial T_e}{\partial x_i}\right) + T_e\frac{\partial u_{ei}}{\partial x_i} + \frac{1}{n_e}\frac{\partial q_i}{\partial x_i} = \sum_\beta \frac{m_e}{2n_e}\int_{\Re^3} |\vec{v} - \vec{u}_e|^2 Q(f_e, f_\beta) d\vec{v} \quad .$$

and comes from the product of the electronic Fokker-Planck equation with $|\vec{v} - \vec{u}_e|^2$ together with an integration in velocity space.

B. The Fokker-Planck Operator

The Fokker-Planck operator, as given by Landau[1] reads:

$$Q(f_\alpha, f_\beta) = \frac{c}{m_\alpha}\frac{\partial}{\partial v_i}\int_{\Re^3} f_\alpha(\vec{v}) f_\beta(\vec{v}\prime) J^{ij}_{\vec{\omega}} \left(\frac{1}{m_\alpha}\frac{\partial Log f_\alpha}{\partial v_j} - \frac{1}{m_\beta}\frac{\partial Log f_\beta}{\partial v'_j}\right) d\vec{v}\prime \qquad (1.2)$$

where,

$$J^{ij}_{\vec{\omega}} = \frac{1}{2}\left(\frac{\delta^{ij}}{|\vec{\omega}|} - \frac{\omega_i\omega_j}{|\vec{\omega}|^3}\right)$$

and,

$$\vec{w} = \vec{v} - \vec{v}\prime$$

$$c = 4\pi e^4 Z_\alpha^2 Z_\beta^2 Log\Lambda_{\alpha\beta} \quad .$$

Rosenbluth[2] has given a less symmetric form of this operator which, to date, has been more frequently used.

$$Q(f_\alpha, f_\beta) = \frac{c}{{m_\alpha}^2}\frac{\partial}{\partial v_i}\left(\frac{\partial f_\alpha}{\partial v_j}\frac{1}{2}\frac{\partial^2 G_\beta}{\partial v_i \partial v_j} - \frac{m_\alpha}{m_\beta} f_\alpha \frac{\partial H_\beta}{\partial v_i}\right) \qquad (1.3)$$

with,

$$G_\beta = \int_{\Re^3} f_\beta |\vec{\omega}| d\vec{v}\prime \qquad (1.4)$$

and,

$$H_\beta = \int_{\Re^3} f_\beta \frac{1}{|\vec{\omega}|} d\vec{v}\prime. \qquad (1.5)$$

We have the following relations between the Rosenbluth's potentials G_β and H_β:

$$\Delta G_\beta = 2H_\beta$$

$$\Delta H_\beta = -4\pi f_\beta \ .$$

Equations (1.2) and (1.3) are equivalent, but for numerical reasons we shall discretize Eq. (1.3).

C. Ion-Electron Interactions

Taking into account the ion-electron interaction, the Fokker-Planck equation for ions is

$$\frac{\partial f_\alpha}{\partial t} + v_i \frac{\partial f_\alpha}{\partial x_i} + \frac{q_\alpha E_i}{m_\alpha} \frac{\partial f_\alpha}{\partial v_i} = \sum_\beta Q(f_\alpha, f_\beta) + Q(f_\alpha, f_e) \quad , \tag{1.6}$$

and for electrons,

$$\frac{\partial f_e}{\partial t} + v_i \frac{\partial f_e}{\partial x_i} + \frac{q_e E_i}{m_e} \frac{\partial f_e}{\partial v_i} = \sum_\beta Q(f_e, f_\beta) + Q(f_e, f_e) \quad . \tag{1.7}$$

1. Exchange Ion-Electron in the Ionic Equation"

According to our approximations, we expand the electronic distribution function as:

$$f_e = f_{e0} + \epsilon f_{e1} \ , \tag{1.8}$$

where f_{e0} is the equilibrium Maxwellian ($Q(f_{e0}, fee) \equiv$ O), and f_{e1} is a perturbation computed from f_{e0}.

f_{e0} reads:

$$f_{e0} = \frac{n_e}{(\frac{2\pi T_e}{m_p})^{\frac{3}{2}}} exp(\frac{-m_p(\vec{c} - \epsilon\vec{u}_e)^2}{2T_e}) \quad .$$

where n_e is the electronic density, $\vec{u}_e$ the mean electronic velocity, and T_e the electronic temperature ; $\vec{c} = \epsilon\vec{v}$ where $\vec{v}$ is an electronic velocity. So $\vec{c}$ is of the order of one ion velocity.

We use this form of f_e in the Eq. (1.6), and keep terms up to first order in ϵ.

After some algebra, the Fokker-Planck equation for f_α reads:

$$\frac{\partial f_\alpha}{\partial t} + v_i \frac{\partial f_\alpha}{\partial x_i} + E_{\alpha i} \frac{\partial f_\alpha}{\partial v_i} = \sum_\beta Q(f_\alpha, f_\beta)$$

$$+ \frac{16\pi^2 e^4 Z_\alpha^2}{m_\alpha^2} \log \Lambda_{\alpha e} \frac{n_e m_e^{\frac{1}{2}} m_\alpha}{3(2\pi T_e)^{\frac{3}{2}}} \frac{\partial}{\partial v_i} \left((v_i - u_{\alpha i}) f_\alpha(\vec{v}) + \frac{T_e}{m_\alpha} \frac{\partial f_\alpha}{\partial v_i}(\vec{v}) \right) \quad , \qquad (1.9)$$

with,

$$\tilde{E}_{\alpha i} = \frac{Z_\alpha}{m_\alpha n_e} \left(-\frac{\partial}{\partial x_i}(n_e T_e) + \sum_\beta m_\beta n_\beta X_{\beta i} \right) - X_{\alpha i} \quad , \qquad (1.10)$$

where,

$$\vec{X}_\alpha = \frac{16\pi^2 e^4 Z_\alpha^2}{m_\alpha^2} \log \Lambda_{\alpha e} \left(\frac{n_e m_e^{1/2} m_\alpha}{3(2\pi T_e)^{3/2}} (\vec{u}_\alpha - \vec{u}_e) + A_\alpha \epsilon \frac{\partial \vec{S}_{e1}(\vec{0})}{\partial \vec{c}} \right) \quad ,$$

and,

$$\vec{S}_{e1}(\vec{c}) = -\frac{1}{4\pi} \int_{\mathcal{R}^3} f_{e1} \frac{1}{|\vec{c} - \vec{c}'|} d\vec{c}' \quad .$$

Notice that with only one ionic species and the quasi-neutrality of the plasma, we obtain:

$$\tilde{E}_{\alpha i} = -\frac{Z_\alpha}{m_\alpha n_e} \frac{\partial}{\partial x_i}(n_e T_e)$$

which is the classical expression of the electric field.

2. Electron Equation

Using (1.8) for f_e, we multiply (1.7) by $\frac{1}{2} m_e (\vec{v} - \vec{u}_e)^2$ and integrate over velocity. After some algebra, the equation for the electronic energy reads:

$$\frac{\partial}{\partial t}\left(\frac{3}{2} n_e T_e\right) + \frac{\partial}{\partial x_i}\left(\frac{3}{2} n_e u_{ei} T_e\right) + \frac{\partial}{\partial x_i}(T_e n_e u_{ei}) + \sum_\beta \tilde{E}_{\beta i} m_\beta n_\beta u_{\beta i} + \frac{\partial q_i}{\partial x_i}$$

$$= \sum_\beta 3 C_{e\beta} n_e n_\beta \frac{(T_\beta - T_e)}{T_e^{\frac{3}{2}}} \qquad (1.11)$$

Here again, we only have retained first order terms in ϵ.

The heat flux $\vec{q}$ is evaluated using the classical Spitzer[3] method, and modified to take into account several ion species.

II. NUMERICAL MODEL

We use a splitting method between the transport part and the collisional part of the Fokker-Planck equation to solve it numerically. The acceleration due to the electric field is naturally included in the collision operator.
We solve successively:

$$\frac{\partial f}{\partial t} + v_i \frac{\partial f}{\partial x_i} = 0 \quad ,$$

then,

$$\frac{\partial f}{\partial t} = \sum_{\beta} Q(f_\alpha, f_\beta) + \text{Acceleration} + \text{Exchange electron-ion.}$$

We then solve the diffusion equation for the electron temperature.

We use a rectangular mesh $[0, V_{\perp max}] \times [-V_{\parallel max}, +V_{\parallel max}]$ in velocity space and an equally spaced set of points for the dimension $[0, R_{max}]$. When the boundary R_{max} moves to R'_{max}, we compute a new set of equally spaced points to discretize $[0, R'_{max}]$.

A. Transport

In plane geometry, the transport part of the equation reads:

$$\frac{\partial f}{\partial t} + v_\parallel \frac{\partial f}{\partial x} = 0$$

and in spherical geometry,

$$\frac{1}{r^2 v_\perp}\Big(\frac{\partial}{\partial t}(f r^2 v_\perp) + \frac{\partial}{\partial r}(v_\parallel (f r^2 v_\perp)) - \frac{\partial}{\partial v_\perp}(\frac{1}{r} v_\perp v_\parallel (f r^2 v_\perp)) + \frac{\partial}{\partial v_\parallel}(\frac{1}{r} v_\perp^2 (f r^2 v_\perp))\Big) = 0 \quad . \tag{2.1}$$

For brevity, we have not reproduced here the terms (1.1) which come from the change of frame. We shall see that our numerical method will naturally include them.

An explicit method[4] gives good results in plane geometry, but shows a drastically decreasing time step at the center of the geometry, coming from a Courant-Friedrichs-Levy condition like:

$$\frac{1}{r} v_\perp v_\parallel \Delta t \leq \Delta v_\perp \quad .$$

Implicit methods come out to be very dispersive. For all these reasons, we have adopted a particle method to solve our transport equation.[5,6,7]

1. Particle Method

We shall consider here only spherical geometry. Plane geometry is straightforward. We replace the function $r^2 v_\perp f$ with a sum of Dirac distributions:

$$r^2 v_\perp f = \sum_p \omega_p J_p(t) r_p^2(t) v_{\perp p}(t) f_p(t) \delta(r - r_p(t)) \delta(v_\perp - v_{\perp p}(t)) \delta(v_\parallel - v_{\parallel p}(t)) \quad .$$

w_p is the quadrature weight in the space $(r, v_\perp, v_\parallel)$, related to the location $(r_p(0), v_{\perp p}(0), v_{\parallel p}(0))$.

Along a characteristic, $J_p(t)$ obeys the equation,

$$\frac{d}{dt} J_p(t) = J_p(t) div \vec{a}$$
$$J_p(0) = 1.$$

$\vec{a}$ has the components $(v_\parallel, -\frac{1}{r} v_\perp v_\parallel, \frac{1}{r} v_\perp^2)$ in the space $(r, v_\perp, v_\parallel)$, and we have $J_p(t) = \frac{r_p(0)}{r_p(t)}$.

Considering $r_p(t) v_{\perp p}(t)$, which is constant along a characteristic, we have

$$\omega_p J_p(t) r_p^2(t) v_{\perp p}(t) f_p(t) = \omega_p r_p(t) r_p(0) v_{\perp p}(t) f_p(t) = \omega_p r_p(0) r_p(0) v_{\perp p}(0) f_p(t).$$

We then deduce $f_p(t)$ from $f_p(0)$ with the particle equation,

$$\frac{d}{dt} (\omega_p J_p(t) r_p^2(t) v_{\perp p}(t) f_p(t)) = 0 \quad .$$

We obtain,

$$r^2 v_\perp f = \sum_p \omega_p r_p^2(0) v_{\perp p}(0) f_p(0) \delta(r - r_p(t)) \delta(v_\perp - v_{\perp p}(t)) \delta(v_\parallel - v_{\parallel p}(t)) \quad .$$

We shall detail the choice of the quadrature points.

Each cell of the initial mesh is divided in $n_{v_\parallel} \times n_{v_\perp} \times n_r$ smaller boxes, where $n_{v_\parallel}$ is the number of particles per cell in the direction $v_\parallel$, $n_{v_\perp}$ in the direction $v_\perp$, and n_r in the direction r. A good choice is $n_{v_\parallel} = n_{v_\perp} = n_r = 2$.

The quadrature weights are the volumes,

$$\omega_p = (\Delta r)_i (\Delta v_\perp)_k (\Delta v_\parallel)_j$$

The quadrature points are chosen in order to make the total weight of a particle p—that is, $\omega_p r_p^2(0) v_{\perp p}(0)$—equal to the volume of the box associated with the particle. This choice forces the location of the particle to be:

$$r_p^2(0) = \frac{r_{i+1}^3 - r_i^3}{3(\Delta r)_i} = \sqrt{\frac{r_i^2 + r_{i+1}^2 + r_i r_{i+1}}{3}}$$

and,

$$v_{\perp p}(0) = \frac{1}{2}\left(v_{\perp k} + v_{\perp k+1}\right) \quad , \quad v_{\parallel p}(0) = \frac{1}{2}\left(v_{\parallel j} + v_{\parallel j+1}\right).$$

We now have to choose $f_p(0)$. A simple but poor choice would be to take f as constant in a cell and to pick its initial value. A more accurate choice is to replace f by a linear expansion in $(r^3, v_\perp^2, v_\parallel)$, the slopes being those of Van-Leer,[4]

$$f = f_M + p_{r^3}(r^3 - r_M^3) + p_{v_\perp^2}(v_\perp^2 - v_{\perp M}^2) + p_{v_\parallel}(v_\parallel - v_{\parallel M} \quad ,$$

where M is the center of the initial cell scaled as $(r^3, v_\perp^2, v_\parallel)$.

We take $f_p(0)$ to be the value of f at the center of the box associated with the particle. This value ensures that the number of particles computed before the transport solution will be the same as if it were computed with a value of f constant in a cell:

$$\sum_p \omega_p r_p^2(0) v_{\perp p}(0) f_p(0) = \sum_M V_M f(M)$$

where V_M is the volume of the initial cell.

The particles are then moved along the characteristics, and f is computed at the end of the time step on each final cell with the relation:

$$f_M = \frac{\sum_p \omega_p r_p^2(0) v_{\perp p}(0)\ f_p(0) \frac{V_{p \cap M}}{V_p}}{V_M} \quad , \tag{2.2}$$

where V_p is some control volume associated to the particle p.

This volume is defined by our choice of a shape (cutoff function) for the particles, which can be either a "step" function or a "hat" function. The location of the shape function is the location of the particle at the end of the time step, $(r_{i+\frac{1}{2}}, v_{\perp k+\frac{1}{2}}, v_{\parallel j+\frac{1}{2}})$ and its width is the cartesian product (r_i, r_{i+1}) x $(v_{\perp k}, v_{\perp k+1}) \times (v_{\parallel j}, v_{\parallel j+1})$.

After projection on the final mesh, the particles are lost and replaced by f, which is the initial condition for collisions.

The above projection only keeps the particle number constant. The advantages of the method are to get rid of the CFL condition, to allow any boundary condition, and to account naturally, when we project the particles on the new grid, for the cumbersome terms coming from the change of frame. Besides, the particles are moved independently, which makes this method a good candidate for parallelization.

B. Collisions

We discretize Rosenbluth's form of the Fokker-Planck collision operator. This choice benefits from the results of a long practice in Fokker-Planck codes. Unfortunately, it has the inconvenience of not fulfilling some conservation properties.

1. Discretization of the Fokker-Planck Operator

We have chosen a 1-D plane or spherical geometry. The distribution functions are function oft, $|\vec{r}|, v_\perp$ and $v_\parallel$ only (not ϕ).

We see that $v_\perp, v_\parallel$ and ϕ are the cylindrical coordinates of the vector $\vec{v}$ in a frame whose axis is perpendicular to the infinite plan in plane geometry, and the segment OM where O is the center of the sphere and M the space point in spherical geometry.

In this frame, the collision operator reads,

$$Q(f_\alpha, f_\beta) = \frac{c}{{m_\alpha}^2} \frac{1}{v_\perp} \frac{\partial}{\partial v_i} v_\perp \left(\frac{\partial f_\alpha}{\partial v_j} \frac{1}{2} \frac{\partial^2 G_\beta}{\partial v_i \partial v_j} - \frac{m_\alpha}{m_\beta} f_\alpha \frac{\partial H_\beta}{\partial v_i} \right) \quad . \tag{2.3}$$

with $v_1 = v_\perp$ and $v_2 = v_\parallel$, and Rosenbluth's potentials,

$$G_\beta(v_\perp, v_\parallel) = \int_0^{+\infty} \int_{-\infty}^{+\infty} \left(\int_0^{2\pi} |\vec{v} - \vec{v}'| d\phi' \right) f_\beta(v'_\perp, v'_\parallel) v'_\perp dv'_\perp dv'_\parallel \tag{2.4}$$

and

$$H_\beta(v_\perp, v_\parallel) = \int_0^{+\infty} \int_{-\infty}^{+\infty} \left(\int_0^{2\pi} \frac{1}{|\vec{v} - \vec{v}'|} d\phi' \right) f_\beta(v'_\perp, v'_\parallel) v'_\perp dv'_\perp dv'_\parallel \quad . \tag{2.5}$$

The azimuth around the axis of symmetry of the vector $\vec{v}'$ is ϕ'.

The collision operator conserves the density n, the total momentum, and the total energy.

For two species, its equilibrium solutions are 2 Maxwellians at the same temperature and the same mean velocity:

$$f_\alpha = \frac{n_\alpha}{(\frac{2\pi T}{m_\alpha})^{\frac{3}{2}}} \exp\left(\frac{-m_\alpha(v_\perp^2 + (v_\parallel - U_\parallel)^2)}{2T}\right)$$

and

$$f_\beta = \frac{n_\beta}{(\frac{2\pi T}{m_\beta})^{\frac{3}{2}}} \exp\left(\frac{-m_\beta(v_\perp^2 + (v_\parallel - U_\parallel)^2)}{2T}\right) \quad .$$

2. Rosenbluth's Potentials

The domain $[0, V_{\perp max}] \times [-V_{\parallel max}, +V_{\parallel max}]$ is discretized with rectangles. We need to know the quantities $\frac{\partial H_\beta}{\partial v_i}$ and $\frac{\partial^2 G_\beta}{\partial v_i \partial v_j}$ for $v_i = v_\parallel, v_j$ = VII and $v_j = v_\perp$ at the center of the segments $\Delta v_\perp$, 3 coefficients, and the same quantities for $v_i = v_\perp$, $v_j = v_\parallel$ and $v_j = v_\perp$ at the center of the segments $\Delta v_\parallel$, 3 more coefficients.

To compute Rosenbluth's potentials, we solve two Poisson equations,

$$\Delta H_\beta = -4\pi f_\beta$$

$$\Delta G_\beta = 2H_\beta$$

We use a finite element method, P_1, on a triangular mesh whose nodes are at the middle of the faces of the initial rectangular mesh. We have inhomogeneous Dirichlet limit conditions at the domain boundaries.

To compute these boundary conditions, we take advantage of the fact that $(\int_0^{2\pi} |\vec{v} - \vec{v}'| d\phi')$ and $(\int_0^{2\pi} \frac{1}{|\vec{v}-\vec{v}'|} d\phi')$ are complete elliptical integrals of the first and second kind* which depend only on $(v_\perp, v_\parallel, \mathrm{Vi}', v_\parallel')$.

These integrals are computed once for $(v_\perp, v_\parallel)$ at the domain boundaries and for $(\mathrm{V_i}', v_\parallel')$ at the center of the inner triangles. H_β and G_β, at the boundaries, are then computed with a simple quadrature using these coefficients, the area of the triangles and the value of f_β at the center of the triangles. The coefficients $\frac{\partial H_\beta}{\partial v_i}$ and $\frac{\partial^2 G_\beta}{\partial v_i \partial v_j}$ are then deduced by means of bicubic spline interpolation.

* M. Abramowitz, I. A. Stegun, Handbook of Math Functions, p 589.

3. Finite Volumes

With the form (1.3) of the collision operator, we assume that f_α is constant within an elementary cell Ω of the velocity domain. After a product by a test function $\psi = 1$ and an integration by parts, we have:

$$\int_\Omega d\vec{v}\psi Q(f_\alpha, f_\beta) = \frac{c}{m_\alpha^2}\int_\Gamma d\vec{v}_\Gamma n_i F_i \quad , \tag{2.6}$$

with,

$$F_i = \frac{\partial f_\alpha}{\partial v_j}\frac{1}{2}\frac{\partial^2 G_\beta}{\partial v_i \partial v_j} - \frac{m_\alpha}{m_\beta} f_\alpha \frac{\partial H_\beta}{\partial v_i} \quad .$$

The boundary Γ is the union of the four faces of the rectangle Ω. We assume the flux F to be constant on each of the faces and compute it at their centers. The coefficients $\frac{\partial^2 G_\beta}{\partial v_i \partial v_j}$ and $\frac{\partial H_\beta}{\partial v_i}$ (2.3) are also assumed to be known at the center of the faces.

Let O be the center of a face, we compute at a time t* (see below):

$$F_i(O) = \frac{df_\alpha}{ds}^{n+1}(O) + a^*(o)\ f_\alpha^{n+1}(O) \ ,$$

with,

$$\frac{dv_j}{ds} = \frac{1}{2} \times \frac{\partial^2 G_\beta^\star}{\partial v_i \partial v_j}(O) \quad ,$$

and,

$$a^\star(O) = -\frac{m_\alpha}{m_\beta} \times \frac{\partial H_\beta^\star}{\partial v_i}(O) \quad .$$

The discretization of $F_i(O)$ is the following:

$$F_i(O) = \left(\frac{f(B) - f(A)}{As}\right) + u^*(0)\ ((1 - \delta)f(A) + \delta f(B)) .$$

A and B are the cross points of the segment of slope $\frac{dv_j}{ds}$ going through O, with one of the 6 horizontal or vertical segments of the polygon whose edges are the centers of the 6 cells whose intersection with that segment is not empty ; *f(A)* and $f(B)$ are linearly interpolated from the values of *f* at the edges of these segments; δ is defined as,

$$\delta = \frac{1}{a^\star(O)\Delta s} + \frac{1}{1 - e^{a^\star(O)\Delta s}} \quad .$$

This ensures that the stationary solution of the continuous equation cancels the discretized flux.[8]

The number of points, involved in the calculation of f_α in one cell, is variable in our scheme, with a maximum of 9 points. If the coefficient of the cross derivatives is null, that is $\frac{\partial^2 G_\beta}{\partial v_i \partial v_j}$ = O for i # j, our scheme reduces to the solution of a Laplacian with a 5 point discretization.

C. Time Discretization

We use an implicit scheme:

$$\frac{1}{\Delta t} f_\alpha^{n+1} V^{n+1} = \frac{1}{\Delta t} f_\alpha^n V^n + \sum_\beta F\left(H(f_\beta^\star), G(f_\beta^\star); f_\alpha^{n+1}\right) \quad .$$

The time centering of the coefficients is either explicit: t* = t_n, centered: t* = $t_{n+1/2}$ or fully implicit: $t^\star = t_{n+1}$.

With a time step of the order of a few τ_{ii}, the explicit centering gives good results.

D. Electrons

We have to solve the electron Eq. (1.11), together with the ion Eqs. (1.9) (one for each ion α).

With the help of our splitting method, we have already solved the transport part and the ion-ion collisions, on the moving grid between t_n et t_{n+1}. That is,

$$\frac{\partial f_\alpha}{\partial t} + v_i \frac{\partial f_\alpha}{\partial x_i} = 0.$$

$$\frac{\partial f_\alpha}{\partial t} = \sum_\beta Q(f_\alpha, f_\beta).$$

Coming from the above computation, f_α is now our initial condition. We shall consider now ion-electron collisions and the acceleration due to the electric field, on the new grid at t_{n+1}.

That is, for each species we solve:

$$\frac{\partial f_\alpha}{\partial t} = \frac{C_{e\alpha} n_e}{T_e^{\frac{3}{2}}} \frac{\partial}{\partial v_i} \left((v_i - u_{\alpha i} - \frac{T_e^{\frac{3}{2}}}{C_{e\alpha} n_e} \tilde{E}_{\alpha i}) f_\alpha(\vec{v}) + \frac{T_e}{m_\alpha} \frac{\partial f_\alpha}{\partial v_i}(\vec{v}) \right) \quad ,$$

where,

$$C_{e\alpha} = \frac{4\sqrt{2\pi} e^4 Z_\alpha^2 m_e^{\frac{1}{2}}}{3 m_\alpha} \log \Lambda_{\alpha e} \quad ,$$

and in axisymmetry,

$$\frac{\partial f_\alpha}{\partial t} = \frac{C_{e\alpha} n_e}{T_e^{\frac{3}{2}}} \left\{ \frac{1}{v_\perp} \frac{\partial}{\partial v_\perp} \left[v_\perp \left(v_\perp f_\alpha(v_\perp, v_\parallel) + \frac{T_e}{m_\alpha} \frac{\partial f_\alpha}{\partial v_\perp}(v_\perp, v_\parallel) \right) \right] + \right.$$
$$\left. \frac{\partial}{\partial v_\parallel} \left[v_\perp \left((v_\parallel - u_{\alpha\parallel} - \frac{T_e^{\frac{3}{2}}}{C_{e\alpha} n_e} \tilde{E}_{\alpha\parallel}) f_\alpha(v_\perp, v_\parallel) + \frac{T_e}{m_\alpha} \frac{\partial f_\alpha}{\partial v_\parallel}(v_\perp, v_\parallel) \right) \right] \right\} \quad .$$

We solve this equation using the same method as with ion-ion collisions, T_e is the electronic temperature at the beginning of the time step. For every species we compute the total energy variation during ion-electron collisions, which after a summation over all the species, gives an approximation to the quantity (see Eq. 1.11):

$$\sum_\beta \tilde{E}_{\beta i} m_\beta n_\beta u_{\beta i} - \sum_\beta 3 C_{e\beta} n_e n_\beta \frac{(T_\beta - T_e)}{T_e^{\frac{3}{2}}} \quad .$$

The source term S of the electronic equation is the opposite sign of this quantity. In this way, we ensure total energy conservation.

So we solve,

$$\frac{\partial}{\partial t}(\frac{3}{2} n_e T_e) + \frac{\partial}{\partial x_i}(\frac{3}{2} n_e u_{ei} T_e) + \frac{\partial}{\partial x_i}(T_e n_e u_{ei}) + \frac{\partial q_i}{\partial x_i} = S \quad ,$$

which, in 1-D plane or spherical geometry reads,

$$\frac{\partial}{\partial t}(\frac{3}{2} n_e T_e) + \frac{1}{r^{igeo}} \frac{\partial}{\partial r} r^{igeo} (\frac{3}{2} n_e u_e T_e + n_e u_e T_e + q)) = S \ ,$$

with $igeo$ = O or 2 in plane or spherical geometry respectively. We solve this equation on the moving grid. To accomplish this, we integrate the above equation between t_n and t_{n+1} on a cell of the moving grid (r_i, r_{i+1}). We get:

$$W_{i+\frac{1}{2}}(t_{n+1}) - W_{i+\frac{1}{2}}(t_n) + \int_{t_n}^{t_{n+1}} dt (r_{i+1}(t)^{igeo} F_{i+1}(t) - r_i(t)^{igeo} F_i(t)) =$$

$$\int_{t_n}^{t_{n+1}} dt \int_{r_i(t)}^{r_{i+1}(t)} S r^{igeo} dr \quad ,$$

with,

$$W_{i+\frac{1}{2}}(t) = \int_{r_i(t)}^{r_{i+1}(t)} \frac{3}{2} n_e T_e r^{igeo} dr \quad ,$$

and,

$$F_i(t) = \left(\frac{3}{2} n_e (u_e - \dot{r}) T_e + n_e u_e T_e + q\right)\left(r_i(t)\right) \quad ,$$

where $\dot{r}$ is the grid velocity.

We assume T_e to be constant within a cell, the flux is computed at t_{n+1} to get rid of time step constraint. The complete scheme reads,

$$\frac{3}{2} n_{i+\frac{1}{2}}^{n+1} T_{i+\frac{1}{2}}^{n+1} V^{n+1} - \frac{3}{2} n_{i+\frac{1}{2}}^{n} T_{i+\frac{1}{2}}^{n} V^{n} =$$

$$-\Delta t\left(r_{i+1}(t_{n+1})^{igeo} F_{i+1}(t_{n+1}) - r_i(t_{n+1})^{igeo} F_i(t_{n+1})\right) + S\Delta t V^{n+1} \quad ,$$

with,

$$V(t) = \int_{r_i(t)}^{r_{i+1}(t)} r^{igeo} dr \quad ,$$

and,

$$F_i(t_{n+1}) = \left(\frac{3}{2} n_i (u_i - \dot{r}_i) T_i^{n+1} + n_i u_i T_i^{n+1} + q_i\right) .$$

u_i, n_i are computed with an interpolation of the centered quantities ; the heat flux $q_i (q = -K(r)\frac{\partial}{\partial r} T^{3.5})$ is computed using:

$$q_i = -cond_i \times (T_{i+\frac{1}{2}}^{3.5} - T_{i-\frac{1}{2}}^{3.5}) \quad ,$$

with,

$$cond_i = \frac{1}{\frac{1}{2}\left(\frac{\Delta r_{i-\frac{1}{2}}}{K_{i-\frac{1}{2}}} + \frac{\Delta r_{i+\frac{1}{2}}}{K_{i+\frac{1}{2}}}\right)} \quad .$$

A linearization is made with respect to the temperature, which means we have to solve a linear system at each Newton iteration with the temperature variation as unknown.

To obtain the boundary conditions, we preset the flux at both edges of the domain. We always set the flux of conduction at R_{min} and R_{max} to zero. With a piston at R_{max}, moving with the velocity V_{pisd} we choose,

$$F_{R_{max}}(t) \quad = n_i V_{pisd} T_{R_{max}} .$$

$T_{R_{max}}$ *is* chosen to be the temperature of the cell located at R_{max} if we want to have a reflective condition equivalent to the ion ones. We can also impose $T_{R_{max}}$ to force a surface temperature. In any case, the term involving $u_i - \dot{r}_i$ is never retained, the grid velocity being the piston velocity.

E. Numerical Test

We present here a test in order to validate the particle solution of our equation. This test[9] is concerned with the simulation of a strong shock, which is difficult to simulate with conventional hydro codes. In order to save computing time, we consider that the plasma is at the Maxwellian equilibrium, and we replace the Fokker-Planck collision term by the BGK one[10] with a small relaxation time compared to the time step of the simulation (see Fig. 1).

The numerical conditions are:

Number of space cells	100
Number of cells in $v_{\|\|}$	48
Number of cells in $v_{\perp}$	24
Number of particles	921600
Time step	0.005
CPU time	548 s
Wallclock	1304 s (5.2% of 8-CPU)
Memory used	25 MWords

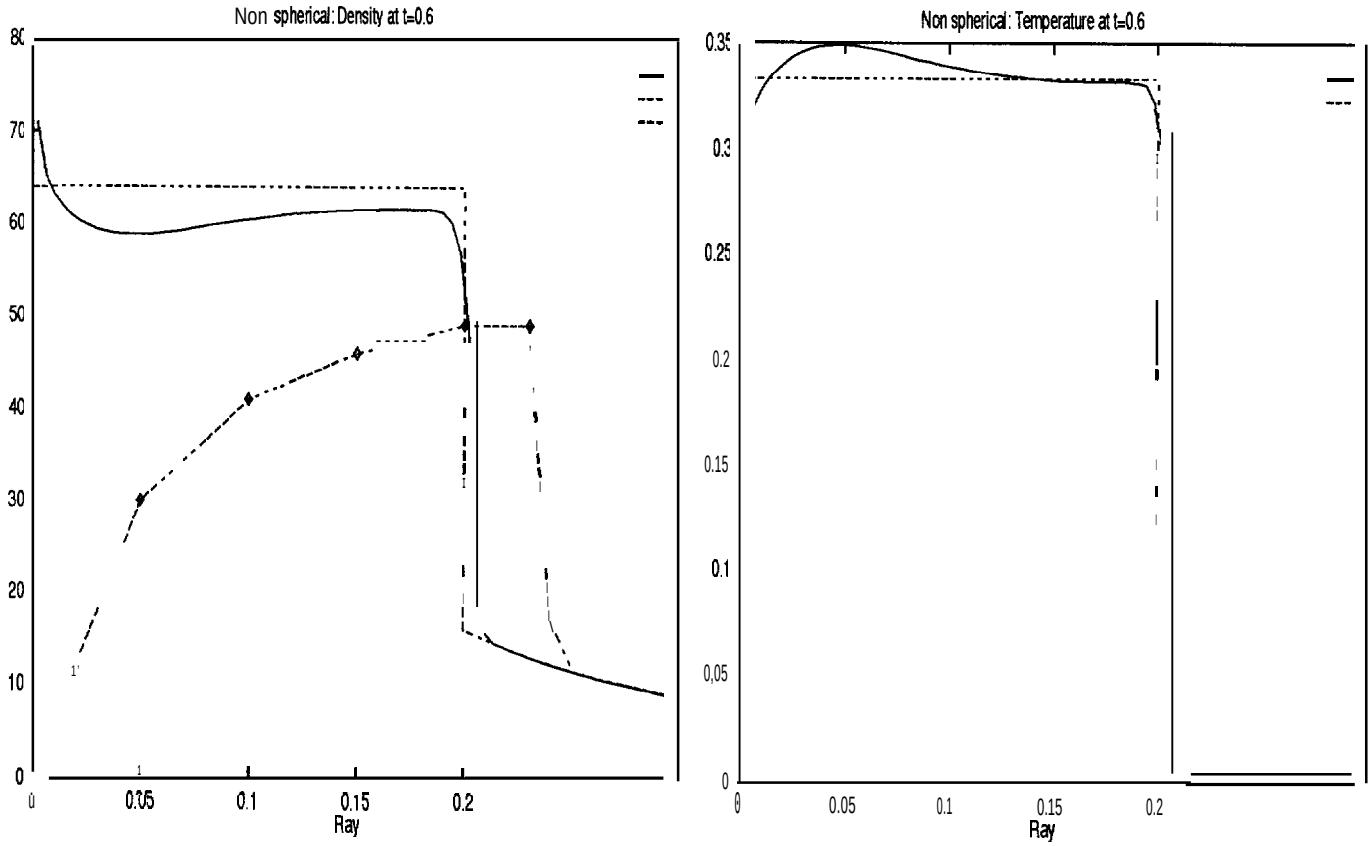


Fig. 1
Density and Temperature profiles at t=0.6
(o Hydro code, —— Kinetic code, ----- Theoretical curve)

The hydro code, based on conventional Richtmyer scheme with standard artificial pseudo viscosity, shows a large drop in the density at the center, resulting in an unphysical jump of the temperature. This jump leads to wrong estimates of the hot spot formation in ICF simulations. The kinetic code gives better results.

We also give the pressure and velocity profiles computed with our kinetic code and compare them to the theoretical profiles (see Fig. 2).

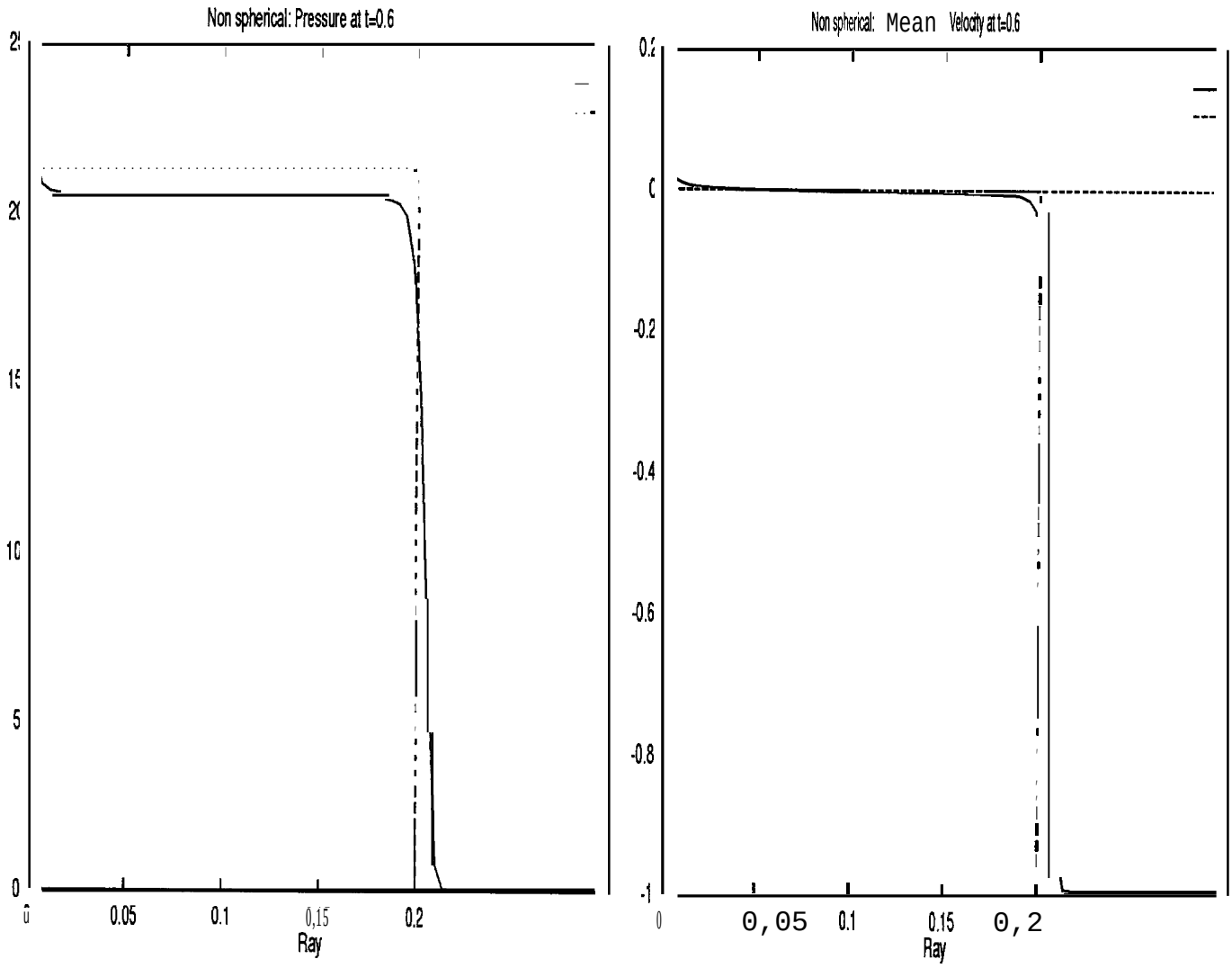


Fig. 2
Pressure and Mean Velocity profiles at t=0.6
(—— Kinetic code, ----- Theoretical curve)

111. PARALLELIZATION

Because we can use a CRAY T3D, at the Los Alamos National Laboratory as well as at the CEA/CEL-V in France, we chose to make our development on the CRAY T3D at the Advanced Computing Laboratory, LANL. However, we want to be able to run our code on a network of workstations as well as on T3D. For this reason we rejected CRAFT (Cray Fortran programming model) and SHMEM (explicit shared memory programing model), and we adopted a message-passing model of parallelization. The free software, Parallel Virtual Machine (PVM), being available on T3D as well as on the network, will be our Message-Passing library.

A. Programming Models

We have adopted a scheme "master-slaves," where a master spawns slaves. On a network of workstations, the master is a specific workstation, chosen from the accessible ones. On the Cray-T3D, the master may be run on the front-end of the T3D, and we speak of this as the "distributed scheme" ; or the master may be a particular PE* on T3D, we then speak of this as the "standalone scheme" or "SPMD " The slaves are always on the T3D. This organization allows us to specialize the master in the 1/0 tasks which are sequential. This method is strictly accurate for the distributed mode but less accurate in our standalone mode, where the master (PE # O) is still a slave.

We show in Fig. 3 the organization of our distributed scheme, where the master program is different from the slave programs, which are all identical. To illustrate our standalone scheme, just remove the "master" part of the Fig. 3. Only the master-slaves communications are shown on this diagram.

We have implemented the two programming models presented above. The communication front-end - PE, and PE - PE are very different on CRAY T3D. We expect the standalone scheme to be the more powerful one.11 We shall see that in our application, which is intended to be as close as possible to a production code, the conclusion of a comparison of the two schemes can be very qualified.

The communications between the front-end and the MPP are the slowest. We minimize the number of those communications, The master spawns **nproc** slaves where **nproc** is the number of PE (2^n on CRAY). Each slave starts its task with an allocation of memory. When all the PEs have completed this allocation, the PE # O sends a message "allocation ok" to the master, which enters an infinite loop and does an 1/0 task each time it receives the order from PE # O. The graphics are then done by the frontend. We shall now detail the work of the slaves.

---

* Processing Element. We shall call "PE" either a T3D elementary processor or a slave process on workstation.

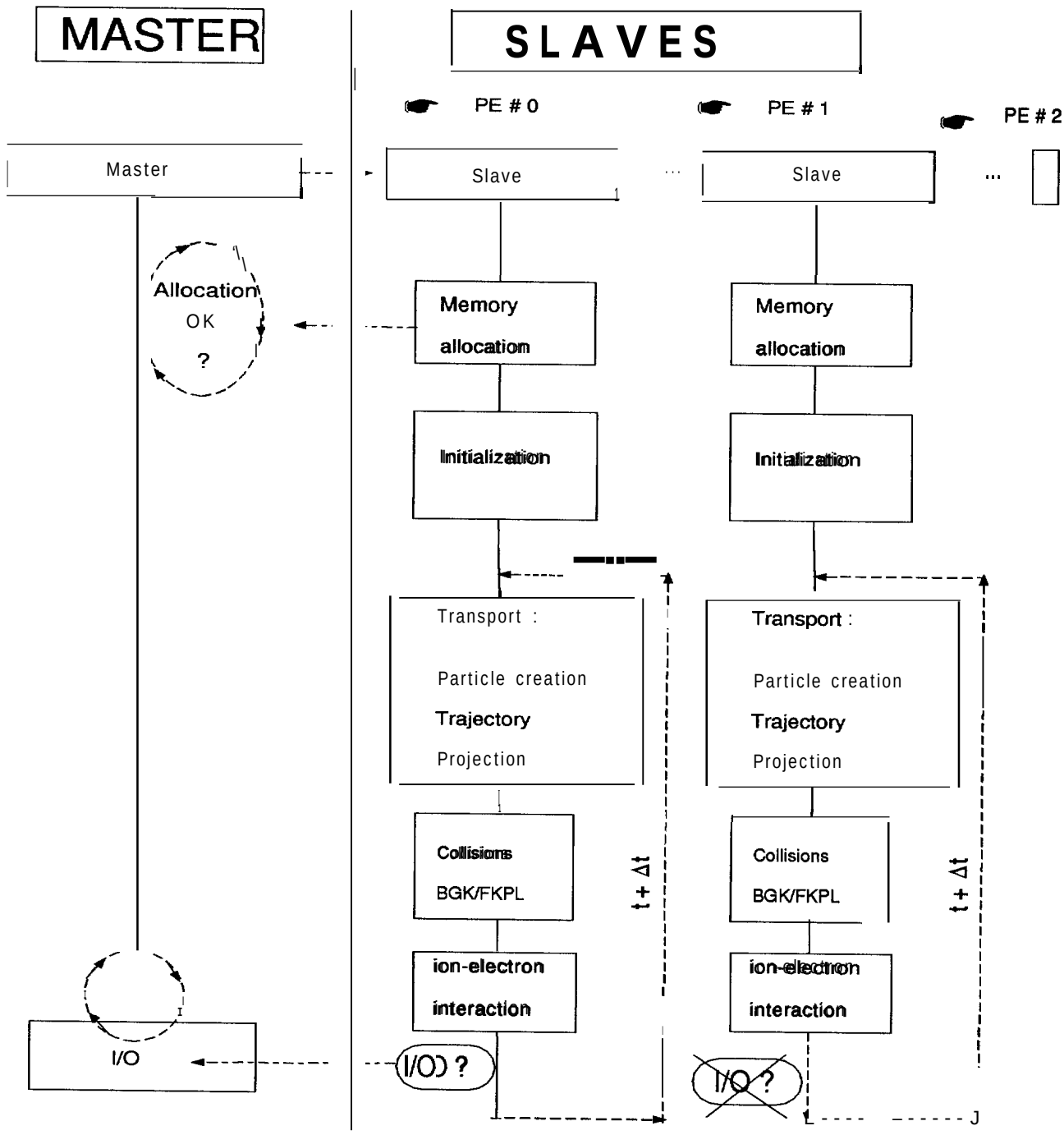


Fig. 3
Organization of a distributed scheme

B. The Slaves

As we already indicated, our splitting method allows several levels of parallelization.

- Transport phase : parallelization with respect to the particles.
- Collision phase : parallelization with respect to the space cells.

A third level of parallelization could be considered when solving the electron-ion interaction. Since this will be done later, we shall not speak of it here.

In each phase, the parallelization is connected to data distribution, which is accomplished by segmenting the data in one or in several directions of phase space. If we divide the spatial mesh into **nproc** segments of n cells each, we can do **nproc** computations in parallel. However, it is necessary to gather the information sometime, that is, to do communications between **nproc** PEs. The efficiency of the algorithm depends on a good balance between the number of communications and the data segmentation.

We have implemented two algorithms hereafter called *Alg 1* and *Alg* 2. The first one, *Alg 1*, limits the number of communications to the detriment of the parallel level. The second one, *Alg 2*, optimizes the data distribution, but increases the number of communications.

Figure 4 shows the unavoidable communications between slaves. The detail of these communications depends on the choice of the algorithm (*Alg 1*, *Alg 2*). The communications coming from an 1/0 task are infrequent. We shall only detail the parallel zones and the resulting communications.

1. Transport Phase

In this phase is computed the variation of the distribution function $f(r, v_{\|}, v_{\perp})$, for a given species due to transport alone (see Eq. 2.1).

This phase includes 3 steps:

(z) Particle creation, which replaces the continuous function f with a population of particles, each of them having a weight, a position r, a velocity $v_{\|}$ and a velocity $v_{\perp}$.

(ii) The trajectory, which is the movement of the particles along the characteristics. During this step, the particles retain their weight, but they change their position and velocities.

(iii) The projection by means of the formula (2.2), rebuilds f in every point of phase space.

Let **npart** be the number of particles with which we have replaced f. Recall that we typically have,

$$\mathbf{npart} = (nptr \times N_r) \times (nptj \times N_{v_{\|}}) \times (nptk \times N_{v_{\perp}}) ,$$

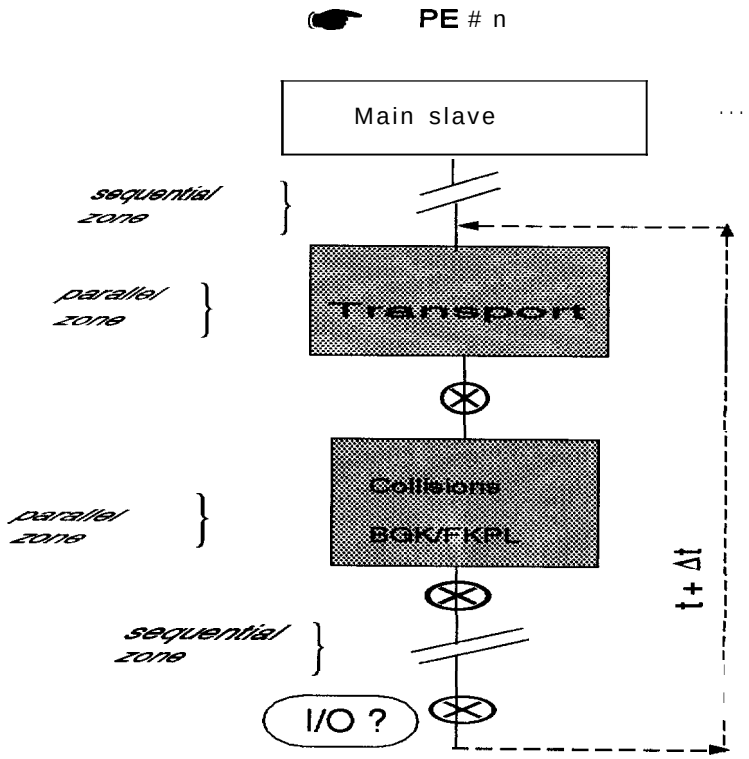


Fig. 4
⊗ Necessary communications between slaves,
and parallel zones (gray areas).

where $nptr$, $nptj$ and $nptk$ are the numbers of particles in each direction $r, v_{\|}$ and $v_{\perp}$.

Step (ii) immediatel$_y$ suggests that we distribute these **npart** particles between the PEs, and simultaneousl$_y$ move these subsets of particles. This distribution is what is done in both algorithms, *Alg 1* and *Alg 2*. So at the step (i) we replace f by pieces on each PE.

We show in Fig. 5 how the particle creation is partitioned on each PE. In the case shown, we have 4 PEs in a "geometry" [2; 2; 1] which means:

[number of PEs in the direction r ; number of PEs in the direction $v_{\|}$;
number of PEs in the direction $v_{\perp}$]

For brevity, we have not represented the direction $v_{\perp}$ on Fig. 5. We can imagine a PE as an "elementary cube. " The union of this set of "elementar$_y$ cubes" is the whole phase space.

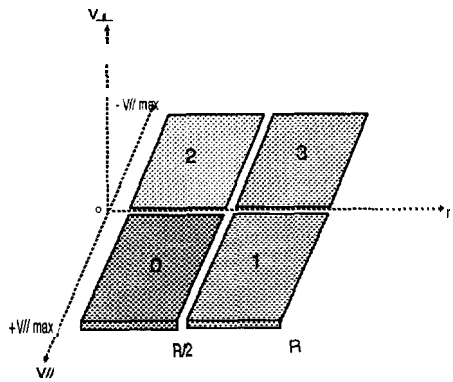


Fig. 5
Piecewise representation of f on PEs.
with subsets of particles.

Figure 5 must be read:

> Particles on PE # O, have positions between O et $R_{max}/2$, velocity $v_{\|}$ between O et $v_{\|_{max}}$, velocity $v_{\perp}$ between (in this case) O et $v_{\perp_{max}}$.
> Particles on PE # 1, have positions between $R_{max}/2$ and R_{max}, velocity $v_{\|}$ between O et $v_{\|_{max}}$, velocity $v_{\perp}$ between O et $v_{\perp_{max}}$.
> Particles on PE # 2, have positions between O et $R_{max}/2$, velocity $v_{\|}$ between $-v_{\|_{max}}$ et 0, velocity $v_{\perp}$ between O et $v_{\perp_{max}}$.
> ...

For any geometry $[N_1; N_2; N_3]$ (the choice is **Up** to the user) where $N_1 \times N_2 \times N_3 =$ **nproc**, the algorithm is:

```
// distribution of the PEs depending on the geometry
// geometry[0]=N1, geometry[1]=N2, geometry[2]=N3
// let PeNum be the number of the executing PE
int     geometry[3],PeCube[3];
int     *ipe , *jpe , *kpe;
int     npartr , npartj , npartk ;

                npartr = ndray X nptr/geometry[0];
                npartj = nvpar X nptj/geometry[1] ;
                npartk = nvperp X nptk/geometry[2] ;
                ipe = new int[nproc] ;
                jpe = new int[nproc] ;
                kpe = new int[nproc] ;

for (i = O ; i < geometry[0] ; i + +)
  for (j = O ; j < geometry[1] ; j + +)
    for (k = O ; k < geometry[2] ; k + +){
                ipe[i + j X geometry[0] + k X geometry[0] X geometry[1] = i ;
                jpe[i + j x geometry[0] + k x geometry[0] x geometry[1] = j ;
                kpe[i + j x geometry[0] + k x geometry[0] x geometry[1] = k ;
}
PeCube[0] = ipe[PeNum] ; PeCube[1] = jpe[PeNum]; PeCube[2] = kpe[PeNum] ;
...
```

Code # 1: Distribution of the phase space between the PEs.

On each PE of number PeNum is defined an array PeCube[3] whose values indicate the "location" of that PE in the discretized phase space, the numbering being that of a 3D Fortran array. Notice, except PE # 0, our numbering has nothing to do with the CRAY partition, it is only relevant to our code.

We then give a particle a position and velocities $v_{\parallel}$ and $v_{\perp}$. The algorithm is the following:

```
c Initialization of the particles parameters
c       We use the array PeCube
        i = 0                          ! particles number
        ir1 = 1 + (PeCube(0) X npartr)
        ir2 = (1 + PeCube(0)) X npartr
        ij1 = 1 + (PeCube(1) X npartj)
        ij2 = (1 + PeCube(1)) X npartj
        ik1 = 1 + (PeCube(2) X npartk)
        ik2 = (1 + PeCube(2)) X npartk
        do ipr = ir1, ir2

        . computes the positions r ∈ [r(ir1), r(ir2)]

          do ipk = ik1 , ik2

          . computes v⊥ ∈ [v⊥(ik1), v⊥(ik2)]

            do ipj = ij1, ij2

            . computes v|| ∈ [v||(ij1), v||(ij2)]

            i = i + 1

            . gives i the position r, velocity v||
            . and velocity v⊥ above computed,
            . and the weight ~ f(r, v||, v⊥)
            enddo
          enddo
        enddo
...
```

Code # 2: Particles definition on each PE.

Step (ii) then follows simultaneously on each PE. Only a PE that knows the boundary of the discretized domain has to deal with boundary conditions.

During this step, particles may "change of PE." More precisely, they may get new positions or new velocities which have been defined on a neighboring PE (in our numbering). We limit the time step in order not to allow a particle to jump a PE in one time step. In other words, a particle may not cross too many cells in one time step, for two reasons:

- We want to limit the number of overlapping cells (see later) between neighboring PEs.
- We consider the particles belonging to the cells close to the boundary the only ones to be searched for "reflection" on a boundary.

During the projection of the particles on the mesh (iii), we rebuild f (see Eq. 2.2) piecewise, each piece coming from the part of the phase space covered by the particles on one PE. To get the complete f, we sum all these contributions, that is we do communications between all of the PEs. It is with respect to these communications and to the part of the phase space known by each PE, that we distinguish the two algorithms, *Alg 1* and *Alg 2*.

2 *Alg 1* and Construction of f

In this algorithm, we want to limit the number of communications between PEs. To achieve this, we define in the sequential steps of the initialization (see Fig. 4) f completely on each PE. In other words, a PE knows the whole mesh. The PEs compute the same thing, but this is not a loss of efficiency, the construction of f being an unavoidable synchronization barrier. The main drawback of this scheme, is its memory consumption. With a large mesh, or a large number of different species, we shall have to use *Alg 2*.

As each PE knows the whole mesh, the rebuilding of f is greatly simplified. There is no real transfer of particles from one PE to another. We only have to sum on PE # O all the contributions to f from every PE, and then send the result to the whole partition.

To have communication with all the PEs together with the maximum effectiveness, we have used an algorithm, whose number of synchronization barriers scales as $\log(\mathbf{nproc})$. This algorithm can be sketched as follows for **nproc** = 16,

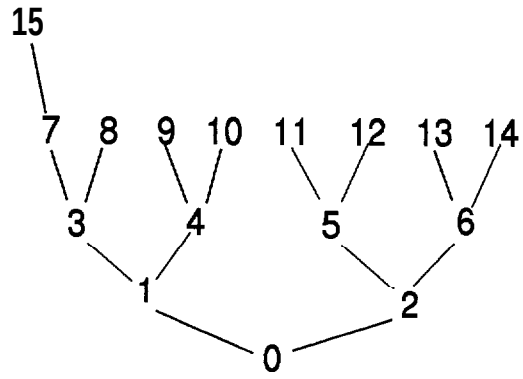


Fig. 6
Communication tree between all the PEs.

The code corresponding to Fig. 6 is very simple. Each PE runs,

```
// defines the "father" that receives the information
// and of the "child" that sends the information
int father ;
int child1 = 2 X PeNum + 1 ;
int child2 = 2 X PeNum + 2 ;

    if ( PeNum% 2 ) {
      father = ( PeNum-1) / 2 ;
    } else{
      father = ( PeNum - 2) / 2 ;
    }

    if ( child1 < nproc ) {
      .wait here to receive the message from child1
      . then add f(PeNum) and f(child1)
    }

    if ( child2 < nproc ) {
      .wait here to receive the message from child2
      . then add f(PeNum) and f(child2)
    }

    if ( PeNum!= O ) {
        send local result to father
    } else{
      .here local result is global result
      . send to all PEs the global result
    }
    if ( PeNum!= O) {
      .wait here to receive the global result
      . sended by the PE with PeNum = O
    }
```

Code # 3 : Communications following Fig. 6.

Here the messages are received by calling a blocking routine which is a natural synchronization barrier. The above algorithm ensures that we always have the maximum number of PEs working at the same time. At the last test reached, the new f is known by every PE, and we can go to the collision part of our equation.

Alg 1 will turn out to be unusable when the phase space is too large. We may also want an optimum splitting of the data on a restricted number of PEs. This is the aim of *Alg 2*.

3. *Alg 2* and Construction of f

In this algorithm, as in *Alg 1*, the **npart** particles are distributed between the **nproc** PEs, but we only define f piecewise on each PE. Given a geometry $[N_1; N_2; N_3]$, we can divide each direction of phase space into the number of PEs in that direction. The r axis is discretized in N_1 segments of N_r/N_1 cells each, the axis $v_\parallel$ in N_2 segments of $N_{v_\parallel}/N_2$ cells each , etc , . .

Although it is theoretically possible to cut out the axis $v_\parallel$ and $v_\perp$, this would greatly complicate the computation of the moments of f, for example, the density:

$$n = 2\pi \int_0^{\infty} \int_{-v_{\parallel max}}^{+v_{\parallel max}} f(r, v_\parallel, v_\perp) v_\perp dv_\perp dv_\parallel \quad .$$

We shall cut out only the axis r in a first step. In other words, the definition of f on one PE is,

$$f\left(r(n_1) \ldots r(n_2), -v_{\parallel max} \ldots + v_{\parallel max}, 0 \ldots v_{\perp max}\right) \quad ,$$

where

$$n_1 = 1 + (PeCube(0) \times N_r/N_1) \ ,$$

and,

$$n_2 = (1 + PeCube(0)) \times N_r/N_1 .$$

The array $PeCube$ is defined same way as in *Alg 1*(see Code # 1).

We modify the diagram of Fig. 5 in order to take into account the partitioning of the axis r for **nproc** = 8.

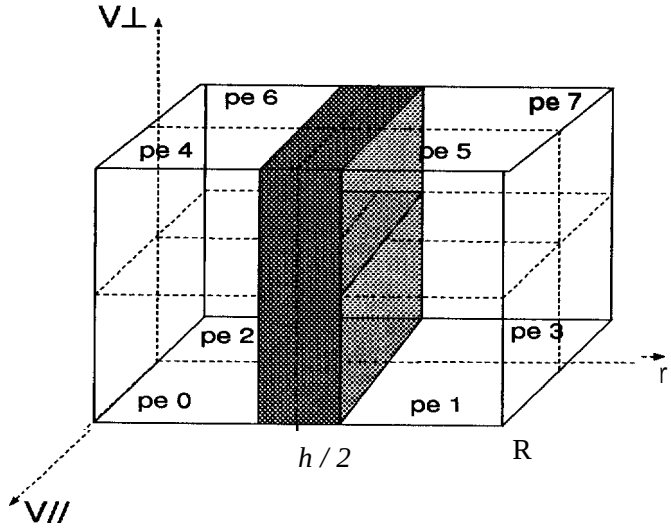


Fig. 7
Portion of the r axis known by each PE
in case of a geometry [2; 2; 2].

In the case of Fig. 7, f is defined from $rmin$ (here O), to $rmax/2$ on PEs # O, 2, 4 and 6. On PEs # 1, 3, 5, and 7, f is defined from $rmax/2$ to $rmax$. As we have not partioned axes $v_{\|}$ and $v_{\perp}$, the space $[v_{\|}, v_{\perp}]$ is known by every PE. However, as in *Alg1*, the creation of the particles is still made on 'tenementary cubes" of phase space. The numbering is the same for the two algorithms *Alg1* and *Alg2* (on Fig. 7, the dashed lines do not represent a division of the phase space; they only show the "elementary cubes" as defined in 3.2.1).

In order to rebuild f after projection (iii), we must sum the contributions of the PEs which cover the same spatial segment (same values of n_1, n_2 above). In the example of Fig. 7, the PEs O, 2, 4 and 6 contribute to the computation of f between $rmin$ and $rmax/2$, the PEs 1, 3, 5 and 7 contribute to the computation of f between $rmax/2$ and $rmax$. The same tree is used for these communicant ions (see Fig. 6) and the same code (see code # 3).

On the CRAY T3D with 256 PEs, the maximum number of PEs which communicate all together is 32, for a geometry [8; 8; 4]. There are 8 groups of PEs doing these communications in parallel ; this number of groups comes from the number of segments (N_1) in the *r* direction. Although the algorithms used for these communications are the same as in *Alg 1*, here the performance is better, the maximum number of PEs we have to make communicate being smaller. However, prior to these communications, other ones have to be done.

Here, during the trajectory, because they may reach positions (and velocity if we had partitioned the velocity axes) which do not belong to the definition of f on a given PE, particles may effectively change PE. To take into account these flights of particles, the cells of neighboring PEs overlap in the direction of the partitioning. These are the gray zone on Fig. 7. The number of overlapping cells depends on the time step of our simulation.

In the example illustrated in Fig. 7, PEs O and 1 are neighbors, as are PEs 2 and 3, PEs 4 and 5, and PEs 6 and 7. The communications are to be done both ways. For instance, PE # O "gives" to PE # 1 those of its particles which have reached a position $r > rmax/2$, and the PE # 1 *"gives"* to the PE # 0 those of its particles which have reached a position r < $rmax/2$. Practically, we do not exchange particles, but the values of f rebuild after projection in the overlapping cells.

The choice of a geometry $[N_1; N_2; N_3]$ is free, providing N_1 x N_2 x N_3 = *nproc*. N_1, N_2, N_3 must be even or equal to one. This free choice allows the adjustment of the partitioning in a given direction.

For any given geometry, the definition of neighboring PEs, and the communications between them, are the following:

```
// communications between right and left neighbors
// voisin[0] is the left neighbor
// voisin[1] is the right neighbor
      int voisin[0] = (PeCube(0) ! = 0) ? (PeNum - 1) : -1 ;
      int   voisin[1] = (PeCube(0) < geometry[0] — 1 ) ? (PeNum + 1) : nproc ;

      if (voisin[0] > -1){
            send to voisin[0] the portion of
            f which belongs to it
            wait here to receive from voisin[0] the portion of
            f which belongs to me
            then add to f(PeNum) the contribution
          - of voisin[0]
      }
      if (voisin[1] < nproc){
          - send to voisin[1] the portion of
          - f which belongs to it
          - wait here to receive from voisin[1] the portion of
          - f which belongs to me
          - then add to f(PeNum) the contribution
          - Of voisin[1]
      }
      ...
```

Code # 4: Communications between overlapping cells.

Boundary conditions, when a temperature, a density, and a mean velocity are to be specified, are taken into account by means of ghost cells defined at the boundaries of the discretized phase space. They do not give rise to extra communication.

4. Collision Phase

We solve here the collision operator (BGK or FKPL). This acts only on the velocity space. Therefore it can be solved simultaneously for each spatial cell.
The information we have on f after the transport phase, depends on the algorithm, *Alg 1* or *Alg 2*, which has been chosen ;

- f is completely known on each PE (*Alg 1*).

We may either solve the collisions for all the cells on every PE or distribute the spatial cells between the PEs. In the first case the PEs, also, compute the same thing. We have a loss of efficiency, but no more communications are needed to do

the rest of the computation (ion-electron interactions). This choice may be justified by consistency with *Alg 1*.

If we choose to distribute the spatial cells between the PEs, we certainly win in efficiency (cpu time), with the maximum efficiency being reached when the number of spatial cells (**ndray**) is smaller than the number of PEs (**nproc**). Then **ndray** PEs solve simultaneously the collision operator on 1 spatial cell. However, we will have to gather the information and have communications on the minimum of $(ndray, nproc)$ PEs. The communication algorithm is in this case strictly identical to the one used to rebuild f after transport in *Alg 1 (see* Fig. 6 and Code # 3).

- f is piecewise known on each PE (*Alg 2*).

In this case, we can solve the collision operator on the spatial segment known by each PE. This does not need any additional communication, but the efficiency reached is not maximum. There are only N_1 PEs, corresponding to the N_1 spatial segments, that share the **ndray** spatial cells. However, for each of those N_1 segments, we can also distribute the spatial cells between the $N_2 \times N_3$ PEs that share the same spatial zone (see Fig. 7). This action leads to a few more communications, but a maximum efficiency may be reached.

Notice: with both *Alg 1* and *Alg 2*, when we distribute the spatial cells between the PEs for the collision operator, we refer to this as: "collision optimization."

5. Diagnostics and 1/0

In the distributed scheme, an 1/0 task sends a message to the master on the frontend. In an SPMD (standalone) scheme, this is the PE # O which does the 1/0 request with apparently no additional communication. However, we shall see in the next section that the execution of an 1/0 task on a T3D PE requires the frontend. Besides, the algorithm *Alg 2* indirectly leads to a few more communications. When we want to compute an energy balance, a particle density, etc ... , the computation of the moments of f will give values only for the cells known by each PE. We shall have to gather that scattered information onto PE # O before any global energy balance, any write, or any draw. Those communications are infrequent, but they are part of the performance decrease.

IV. PERFORMANCE MEASUREMENTS

We shall make our measurements on the CRAY T3D. The interesting time is the elapsed time, which is the time a job takes to return to the user. For any "real" user this time is the only relevant one. In the elapsed time, we shall include the cpu time and the communication time between the PEs on MPP. The communication time between the front-end and the MPP depends on the load of the front-end, which is not in our control. Most of our routines are written in C++, so we cannot

use "MPP Apprentice performance tool"* cd which does not yet support C++. Our measurements will be done using the well known "call second." We have fewer details, but still enough accuracy.

Before entering into details of measurements, we may initially ask the question: did we gain in doing the parallelization of our code? We may partially answer that question by doing two runs, with one and two PEs, on the front-end (YMP) of the T3D. Here only the cpu time is relevant and will tell if our code parallelizes or not.

Computation parameters. *Alg 1* with collisions optimization

Number of spatial cells : 40
Number of cells in $v_{\|}$: 48
Number of cells in $v_{\perp}$: 24
Number of particles : 368640
Number of iterations : 40

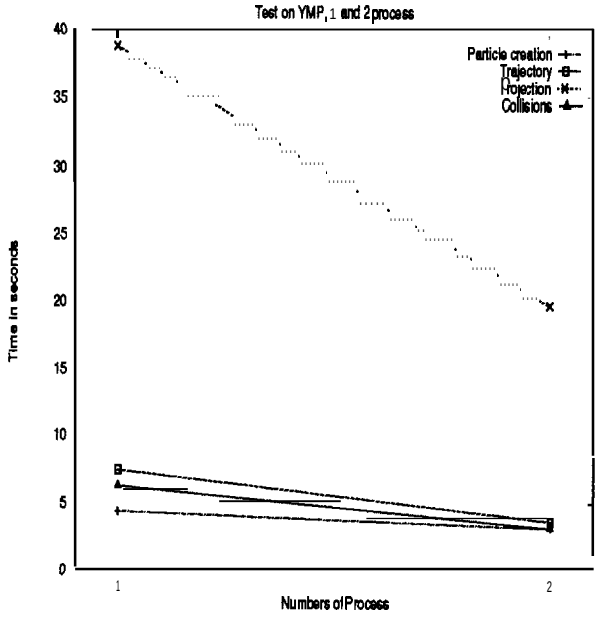


Fig. 8

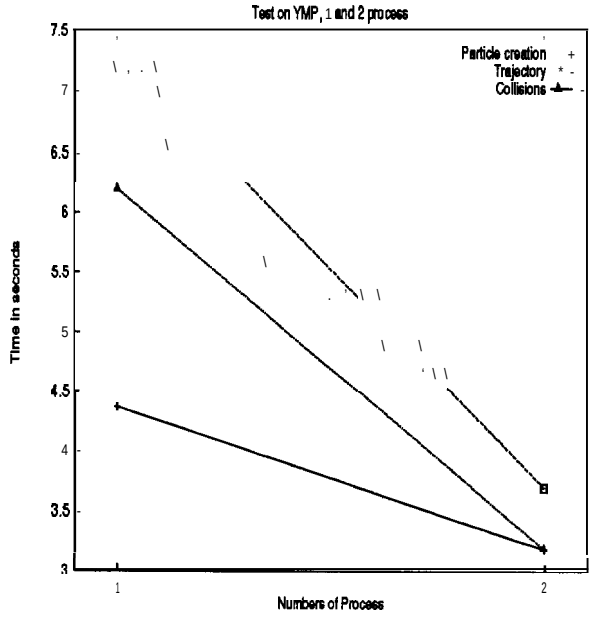


Fig. 9

We show in Fig. 8 the cpu times for the collisions and the three steps of the transport (Fig. 9 is a zoom of Fig. 8 for the three bottom curves).

When going from one process to two processes, apart from the particle creation, the cpu times have been divided by a factor close to two, which is very good. To explain the result of the particles creation, we must keep in mind that most of this step is concerned with the computation of the Van-Leer's slopes, which in *Alg 1* are computed on the whole mesh no matter what the number of PEs is. In the

* CrayTools 1.2 for MPP.

two other steps of the transport, the computation is made on the particles in a subset whose number is divided by a factor of two when the number of PEs is multiplied by a factor of two. In the collision step, the PEs share the spatial mesh. We conclude that our scheme parallelizes very well.

A. SPMD Scheme

In this scheme, there is no explicit communication YMP-T3D. These communications are the least efficient, so we can expect the best performances from this scheme compared to the distributed one (see). Our tests lead to opposite conclusions. In our application, we keep the 1/0 tasks, done here by the PE # O. Unfortunately, to complete an 1/0 request on T3D an external agent (running on the YMP front-end) is involved, and we cannot avoid the load of the front-end. The SPMD scheme appears to be very sensitive to the load of the YMP. The performance measurements turn out to be very difficult to analyse.

Moreover, PVM is not the same on T3D and on YMP. Without a "PVM daemon" (case of the SPMD scheme), the control of the buffers seems different, and we faced a lot of "PVM out of resources.)' In other words, some computations were impossible with this scheme. We shall not present measurements with this scheme.

B. Distributed Scheme

Every test will be done with the following parameters:

Number of spatial cells: 128
Number of cells in $v_{\|}$: 48
Number of cells in $v_{\perp}$: 24
Number of particles : 1179648
Number of iterations : 1

We show in Fig. 10 the total elapsed time for the two algorithms *Alg 1* and *Alg 2* in every option. The maximum speed-up is 13.5. It is obtained with *Alg 2*, optimization of the collisions and with 256 PEs.

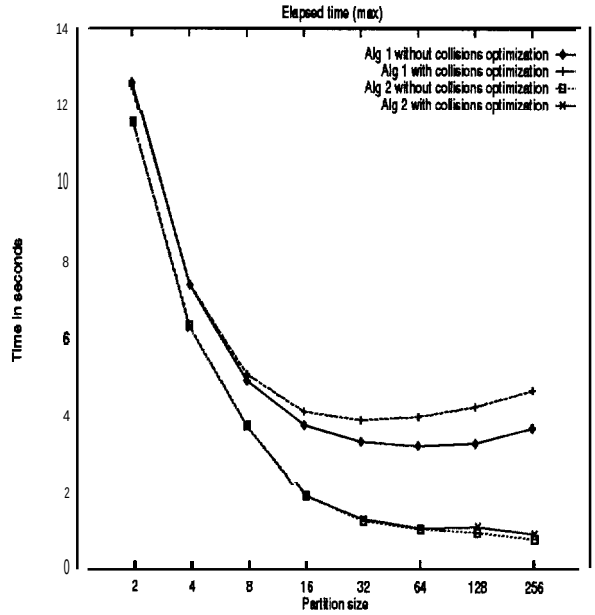


Fig. 10

As we have already mentioned, a message-passing method must find a good balance between the number of communications and the data distribution. The growth of the elapsed time (Fig. 10 *Alg 1*) for **nproc** >32, is due to the increasing number of communications, as can be seen on the Figs. 11 and 12.

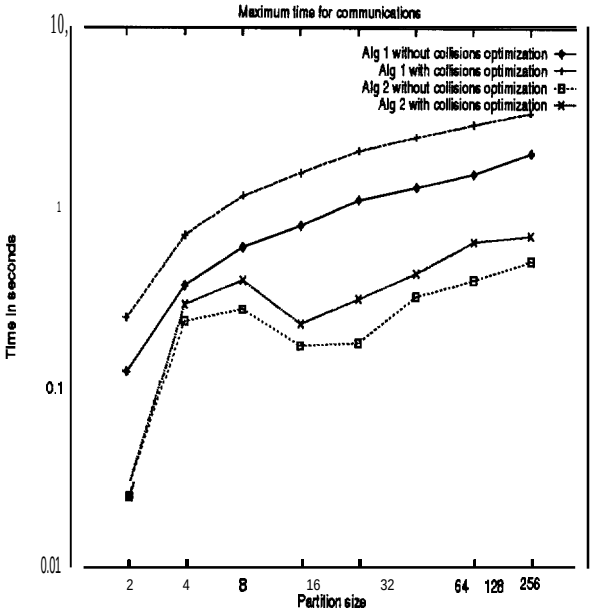


Fig. 11

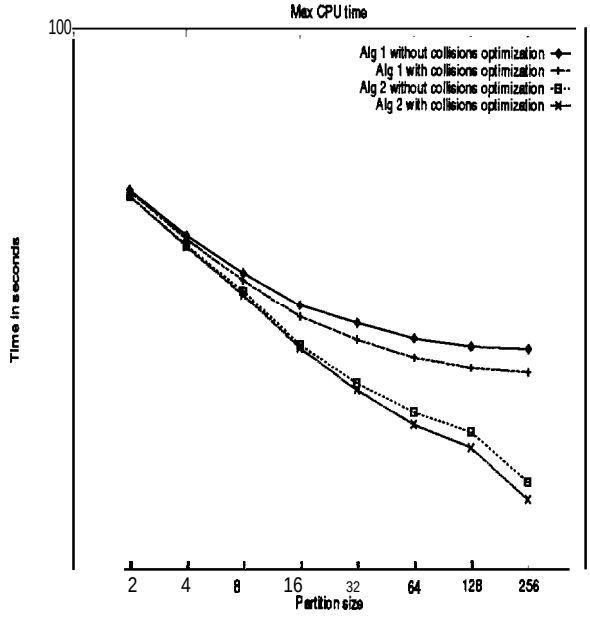


Fig. 12

The cpu times are continuously decreasing with the number of PEs (Fig. 12). However the growth of the communication time (Fig. 11) is enough to reverse the curves of elapsed time (Fig. 10). The location of this minimum depends on

the efficiency of the sofware used (here PVM) as well as on the computer. This minimum may certainly be shifted with the use of shmem_put or shmem_get on CRAY T3D.

We compare now the performances of the algorithms *Alg 1* and *Alg 2*.

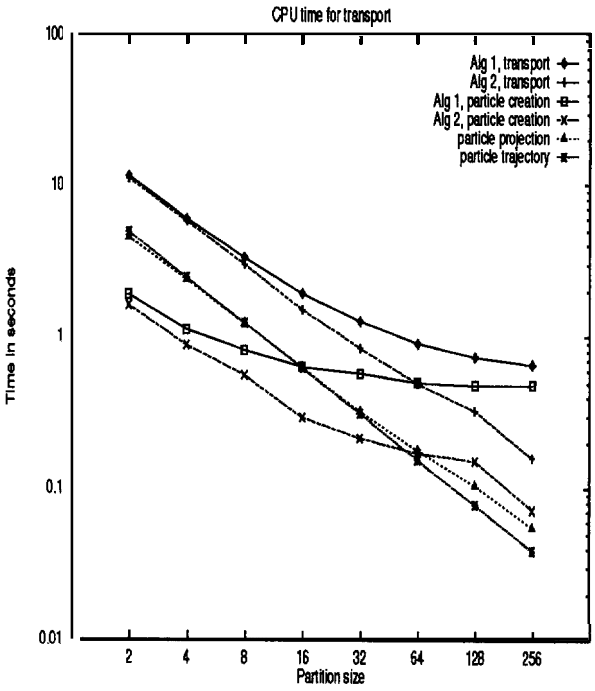


Fig. 13

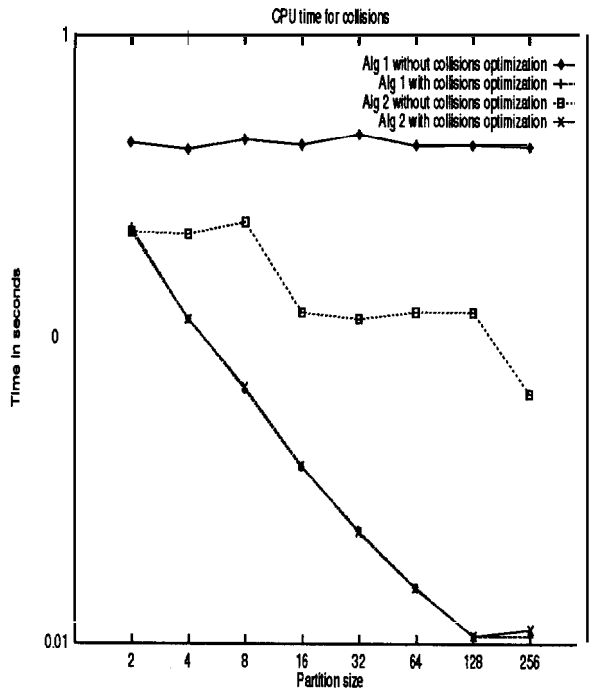


Fig. 14

Figure 13 details the three steps of the transport. The decreasing (equal for *Alg 1* and *Alg* 2), of the trajectory time and the projection time are very good. We find again the dependence of the particle creation on the Van-Leer slopes computation. *Alg 2* shows the best results; the spatial mesh is shared between the N_1 spatial segments related to the geometr$_y$ IN$_1$; N_2; N_3]. The change in the slope of the particle creation curve is due to a change of this geometry ; N_1 = 2 to 8 processors, N_1 = 4 for 16 processors.

We find also the best performances with *Alg 2* (see Fig. 14), for the resolution of the collisions. The time for collision does not depend on the number of PEs in *Alg 1* without optimization. As we expect, the best results are obtained with optimization of the collisions, whatever the algorithm. The maximun efficiency is reached when the number of PEs is equal to or greater than the number of spatial cells.

We show in Fig. 15 the efficiency of the algorithm of communication between every processor as related to the tree of Fig. 6.

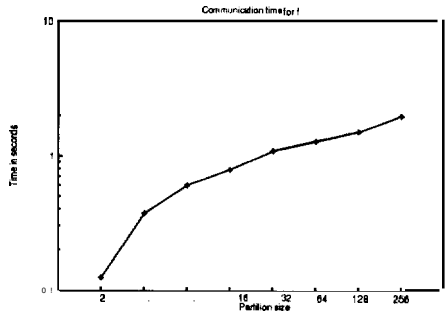


Fig. 15
Effectiveness of the communications
between every PEs.

The mean slope for **nproc** >2 is close to log 2 as we expected.

C. Scalability

We want to study here the variation of the elapsed time versus the size of the problem. More precisely, does the variation of elapsed time stay small when we double both the size of the problem and the number of processors, keeping constant the ratio of cells versus PEs ?

We do our tests with *Alg 2* and the optimization of the collisions, the number of spatial cells being successively: 32, 64, 128, 256. In the same time the number of processors is 8, 16, 32, 64, everything else being kept constant.

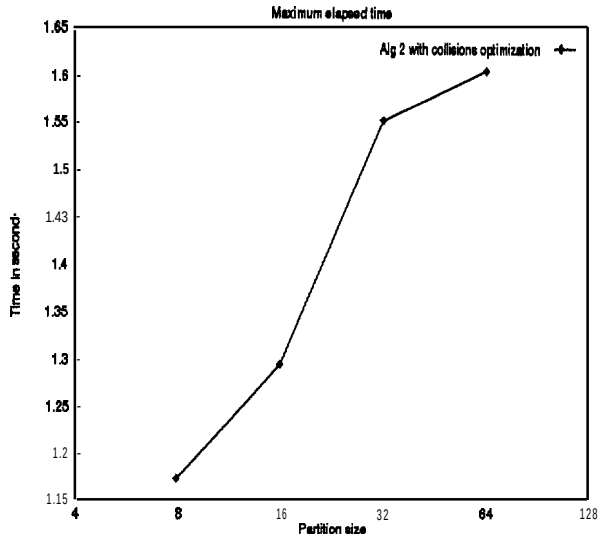


Fig. 16
Elapsed time variation

This makes three tests at very different scales. A 32 space cells test produces a small mesh ; a 256 space cells test is too big to run on one YMP.

Going (see Fig. 16) from 8PEs/32 cells to 16 PEs/64 cells increases the elapsed time by a factor of 1.10. Going from 16 PEs/64 cells to 32 PEs/128 cells increases the elapsed time by a factor of 1.19, and going from 32 PEs/128 cells to 64PEs/256 cells, increases the elapsed time by a factor of 1.03. We see a rather small variation in the elapsed times, and a good stability over a large scale of problem. In the case of 64 PEs and 256 cells, the CPU time *is* of the order of 1 second per PE and the elapsed time of 1.6 ; this makes up the speed to the order of 40, allowing us to conclude that a good scalability is obtained with respect to a non-fixed size problem. Besides, as shown in the two following figures, the variation is mainly due to the communications time,

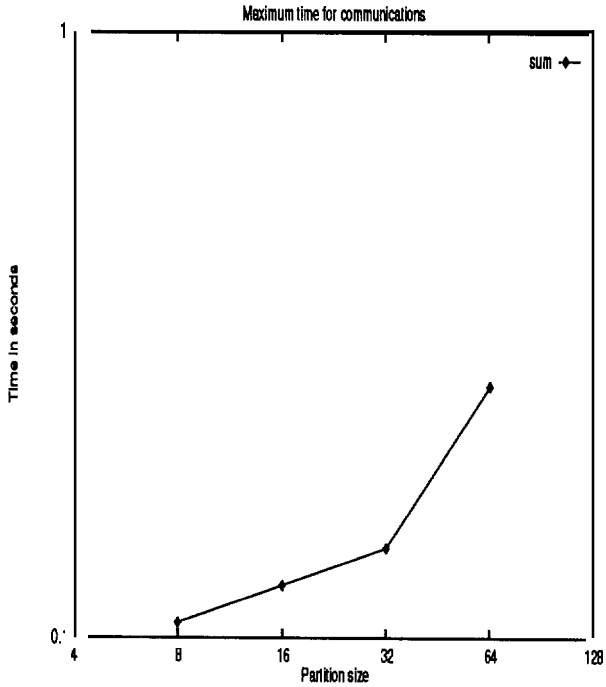


Fig. 17

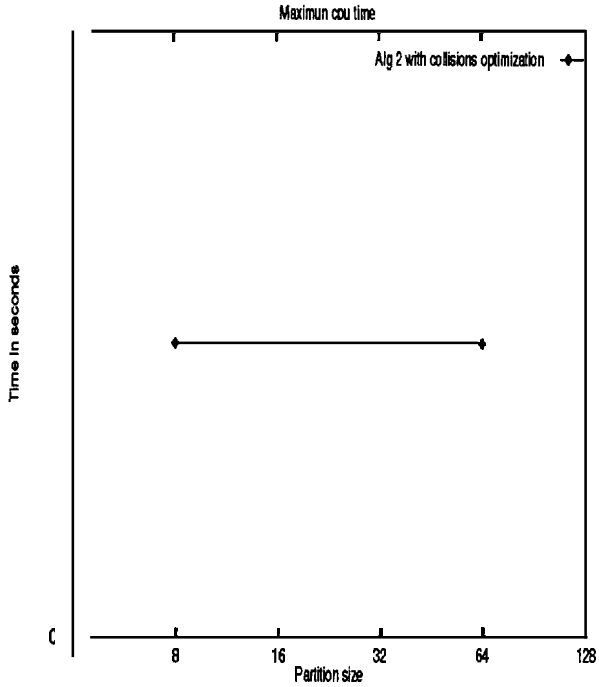


Fig. 18

V. CONCLUSION

We have designed a numerical algorithm to solve a multi-species ion Fokker-Planck equation. This algorithm gives, in spherical geometry, much better results than conventional hydro codes. However, the memory size and the cp time associated with kinetics simulation need the use of MPP computers. We have shown in this work how to parallelize our algorithm. Although still in progress, the preliminary results are very promising, and encourage the porting of our code to a more powerful computer with faster data transfer than the CRAY-T3D.

ACKNOWLEDGMENTS

One of the authors, Dominique Deck, wants to thank Ray Juzaitis, X–Division Director, and Raymond Alcouffe, Transport Methods Group, who made this work possible at the Los Alamos National Laboratory. This work is also deeply indebted to The Advanced Computing Laboratory of the Los Alamos National Laboratory. This work was supported by DGA/DRET under grant 94-1127/A000/DRET/DS/SR.

REFERENCES

1. L. D. Landau, ZhETF (J. Exptl. Theoret. Phys. USSR), 7, 203 (1937).

2. Rosenbluth, MacDonald, and Judd, Phys. Rev., 107, 1 (1957).

3. L. Spitzer, R. Harm, Phys. Rev., 89, 977 (1953).

4. B. Van Leer, Towards the ultimate conservative difference scheme II. J. comput. Phys. 14,361 (1974).

5. J. U. Brackbill, Numerical Methods for charged Particle Transport, Summer School of GDR SPARCH, CNRS, Oléron Sep 1993, France.

6. P. A Raviart, Lecture Notes in Math. Vol. 1127 (Springer-Verlag, 1985).

7. S. Mas-Gallic, Transp. Theory. Stat. Physics, 16, 855 (1987).

8. J. S. Chang, G. Cooper, J. Comput. Phys., 6 (1970).

9. W. F. Noh, J. comput. phys 72, 78 (1978).

10. P. L. Bhatnagar, E. P. Gross, and M. Krook, Phys. Rev., 94, 511 (1954).

11. S. Booth, A. Simpson, and C. Rough, Notes on porting PVM codes to the T3D, EPCC, August 19, 1994.

Los Alamos
NATIONAL LABORATORY
Los Alamos, New Mexico 87545